Hash etudes in C

Mad Hatter: *You're not the same as you were before. You were much more muchier. You've lost your muchness.*

Alice: My "muchness"?

Mad Hatter: *In there. Something's missing.*

```
FNV1A_Hash_WHIZ: 00507-004a0= 103bytes(decimal)
HashFNV1A_Nefertiti: 00576-00510= 102bytes(decimal)
FNV1A_Hash_Peregrine: 00897-00810= 135bytes(decimal)
FNV1A_Hash_Smaragd: 00916-008a0= 118bytes(decimal)
Hash_Sixtinsensitive_Boosted: 003f1-002c0= 305bytes(decimal)
Hash_Sixtinsensitive: 00496-00400= 150bytes(decimal)
HashAlfalfa_QWORD: 00681-00580= 257bytes(decimal)
HashAlfalfa_DWORD: 00710-00690= 128bytes(decimal)
HashAlfalfa_HALF: 007af-00720= 143bytes(decimal)
HashAlfalfa: 00802-007b0= 82bytes(decimal)
FNV1A_Hash_Jester: 00524-004a0= 132bytes(decimal)
FNV1A_Hash_Jesteress: 00521-004a0= 129bytes(decimal)
HashAlfalfa_Rollick: 009b6-00930= 134bytes(decimal)
?FNV1A_Hash_WHIZ@@YAIPBDK@Z PROC                        ; FNV1A_Hash_WHIZ
; 922  : const UINT PRIME = 709607;//15607;
; 923  :
; 924  : //Whiz 107:   10517484  10506006  10516230  10521979  10498682|  10498682 [2162779]
; 925  : //Whiz 113:   10639939  10497081  10501382  10491759  10509496|  10491759 [2145878] !
; 926  : //Whiz 167:   10642142  10494859  10498245  10546216  10514032|  10494859 [2153301]
; 927  : //Whiz 197:   10640694  10494344  10481432  10485134  10514380|  10481432 [2155048]
; 928  : //Whiz 607:   10533478  10562420  10607113  10543554  10563058|  10533478 [2156392]
; 929  : //Whiz 1607:   10702972  10581596  10605821  10620855  10576937|  10576937 [2189360]
; 930  : //Whiz 1777:   10629381  10531970  10538870  10537019  10565007|  10531970 [2164572]
; 931  : //Whiz 2087:   10615378  10541706  10517062  10555783  10515463|  10515463 [2152011]
; 932  : //Whiz 2357:   10642049  10552602  10584649  10490162  10524046|  10490162 [2145307]
; 933  : //Whiz 15607:   10701001  10545813  10513176  10509870  10520269|  10509870 [2138234] !
; 934  : //Whiz 16607:   10665426  10542887  10547560  10548654  10562791|  10542887 [2148125]
; 935  : //Whiz 37607:   10685093  10523184  10511143  10517122  10525008|  10511143 [2142681]
; 936  : //Whiz 70607:   10712369  10537747  10536670  10542325  10516454|  10516454 [2139261]
; 937  : //Whiz 709607: [2138053]
; 938  :
; 939  : UINT hash32 = 2166136261;
; 940  : const char *p = str;
; 941  :
; 942  : for(; wrdlen >= sizeof(DWORD); wrdlen -= sizeof(DWORD), p += sizeof(DWORD)) {
  004a0  8b 54 24 08        mov      edx, DWORD PTR _wrdlen$[esp-4]
  004a4  8b 44 24 04        mov      eax, DWORD PTR _str$[esp-4]
  004a8  56                 push     esi
  004a9  b9 c5 9d 1c 81     mov      ecx, -2128831035 ; 811c9dc5H
  004ae  83 fa 04  cmp      edx, 4
  004b1  72 25              jb       SHORT $LN3@FNV1A_Hash
  004b3  8b f2              mov      esi, edx
  004b5  c1 ee 02  shr      esi, 2
  004b8  57                 push     edi
  004b9  8d a4 24 00 00
         00 00              npad     7
$LL5@FNV1A_Hash:
; 943  :          hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
  004c0  8b 38              mov      edi, DWORD PTR [eax]
  004c2  33 f9              xor      edi, ecx
  004c4  69 ff e7 d3 0a
         00                 imul     edi, 709607              ; 000ad3e7H
  004ca  83 ea 04  sub      edx, 4
  004cd  83 c0 04  add      eax, 4
  004d0  83 ee 01  sub      esi, 1
  004d3  8b cf              mov      ecx, edi
  004d5  75 e9              jne      SHORT $LL5@FNV1A_Hash
  004d7  5f                 pop      edi
$LN3@FNV1A_Hash:
; 944  : }
; 945  : if (wrdlen & sizeof(WORD)) {
  004d8  f6 c2 02  test     dl, 2
```

```
  004db  74 10              je       SHORT $LN2@FNV1A_Hash
; 946  :          hash32 = (hash32 ^ *(WORD*)p) * PRIME;
  004dd  0f b7 30  movzx    esi, WORD PTR [eax]
  004e0  33 f1              xor      esi, ecx
  004e2  69 f6 e7 d3 0a
         00                 imul     esi, 709607               ; 000ad3e7H
  004e8  8b ce              mov      ecx, esi
; 947  :          p += sizeof(WORD);
  004ea  83 c0 02  add      eax, 2
$LN2@FNV1A_Hash:
  004ed  5e                 pop      esi
; 948  : }
; 949  : if (wrdlen & 1)
  004ee  f6 c2 01  test     dl, 1
  004f1  74 0d              je       SHORT $LN1@FNV1A_Hash
; 950  :          hash32 = (hash32 ^ *p) * PRIME;
  004f3  0f be 00  movsx    eax, BYTE PTR [eax]
  004f6  33 c1              xor      eax, ecx
  004f8  69 c0 e7 d3 0a
         00                 imul     eax, 709607               ; 000ad3e7H
  004fe  8b c8              mov      ecx, eax
$LN1@FNV1A_Hash:
; 951  :
; 952  : return hash32 ^ (hash32 >> 16);
  00500  8b c1              mov      eax, ecx
  00502  c1 e8 10  shr      eax, 16                   ; 00000010H
  00505  33 c1              xor      eax, ecx
; 953  : }
  00507  c3                 ret      0
?FNV1A_Hash_WHIZ@@YAIPBDK@Z ENDP                      ; FNV1A_Hash_WHIZ
?HashFNV1A_Nefertiti@@YAIPBDK@Z PROC                  ; HashFNV1A_Nefertiti
; 862  : //const UINT PRIME = 31;
; 863  : UINT hash = 2166136261;
; 864  : const CHAR * p = str;
  00510  8b 54 24 04        mov      edx, DWORD PTR _str$[esp-4]
  00514  53                 push     ebx
; 865  : /*
; 866  : // Reduce the number of multiplications by unrolling the loop
; 867  : for (SIZE_T ndwords = wrdlen / sizeof(DWORD); ndwords; --ndwords) {
; 868  : //hash = (hash ^ *(DWORD*)p) * PRIME;
; 869  : hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);
; 870  :
; 871  : p += sizeof(DWORD);
; 872  : }
; 873  : */
; 874  : for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
  00515  8b 5c 24 0c        mov      ebx, DWORD PTR _wrdlen$[esp]
  00519  b9 c5 9d 1c 81     mov      ecx, -2128831035 ; 811c9dc5H
  0051e  83 fb 04  cmp      ebx, 4
  00521  72 24              jb       SHORT $LN3@HashFNV1A_
  00523  56                 push     esi
  00524  8b f3              mov      esi, ebx
  00526  c1 ee 02  shr      esi, 2
  00529  8d a4 24 00 00
         00 00              npad     7
$LL5@HashFNV1A_:
; 875  :          hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);
  00530  8b 02              mov      eax, DWORD PTR [edx]
  00532  33 c1              xor      eax, ecx
  00534  8b c8              mov      ecx, eax
  00536  c1 e1 05  shl      ecx, 5
  00539  2b c8              sub      ecx, eax
```

```
  0053b  83 eb 04  sub      ebx, 4
  0053e  83 c2 04  add      edx, 4
  00541  83 ee 01  sub      esi, 1
  00544  75 ea              jne      SHORT $LL5@HashFNV1A_
  00546  5e                 pop      esi
$LN3@HashFNV1A_:
; 876  : }
; 877  :
; 878  : // Process the remaining bytes
; 879  : /*
; 880  : for (SIZE_T i = 0; i < (wrdlen & (sizeof(DWORD) - 1)); i++) {
; 881  : //hash = (hash ^ *p++) * PRIME;
; 882  : hash = ((hash ^ *p)<<5) - (hash ^ *p);
; 883  : p++;
; 884  : }
; 885  : */
; 886  : if (wrdlen & 2) {
  00547  f6 c3 02  test     bl, 2
  0054a  74 0f              je       SHORT $LN2@HashFNV1A_
; 887  :          hash = ((hash ^ *(WORD*)p)<<5) - (hash ^ *(WORD*)p);
  0054c  0f b7 02  movzx    eax, WORD PTR [edx]
  0054f  33 c1              xor      eax, ecx
  00551  8b c8              mov      ecx, eax
  00553  c1 e1 05  shl      ecx, 5
  00556  2b c8              sub      ecx, eax
; 888  :          p++;p++;
  00558  83 c2 02  add      edx, 2
$LN2@HashFNV1A_:
; 889  : }
; 890  : if (wrdlen & 1)
  0055b  f6 c3 01  test     bl, 1
  0055e  5b                 pop      ebx
  0055f  74 0e              je       SHORT $LN1@HashFNV1A_
; 891  :          hash = ((hash ^ *p)<<5) - (hash ^ *p);
  00561  0f be 02  movsx    eax, BYTE PTR [edx]
  00564  33 c1              xor      eax, ecx
  00566  8b d0              mov      edx, eax
  00568  c1 e2 05  shl      edx, 5
  0056b  2b d0              sub      edx, eax
  0056d  8b ca              mov      ecx, edx
$LN1@HashFNV1A_:
; 892  :
; 893  : return (hash>>16) ^ hash;
  0056f  8b c1              mov      eax, ecx
  00571  c1 e8 10  shr      eax, 16                 ; 00000010H
  00574  33 c1              xor      eax, ecx
; 894  : }
  00576  c3                 ret      0
?HashFNV1A_Nefertiti@@YAIPBDK@Z ENDP                ; HashFNV1A_Nefertiti
?FNV1A_Hash_Peregrine@@YAIPBDK@Z PROC                       ; FNV1A_Hash_Peregrine
; 766  : {
  00810  55                 push     ebp
  00811  8b ec              mov      ebp, esp
  00813  83 e4 f8  and      esp, -8                 ; fffffff8H
  00816  51                 push     ecx
; 767  : const UINT PRIME = 709607;//15607;
; 768  :
; 769  : //Peregrine 1607: [2128779]
; 770  : //Peregrine 15607: [2108149]
; 771  : //Peregrine 16607: [2110112]
; 772  : //Peregrine 709607: [2104754]
; 773  :
```

```
; 774  : UINT hash32 = 2166136261;
; 775  : const char *p = str;
  00817  8b 4d 08  mov      ecx, DWORD PTR _str$[ebp]
  0081a  53                push     ebx
; 776  : UINT64 OneAccess;
; 777  :
; 778  : for(; wrdlen >= sizeof(UINT64); wrdlen -= sizeof(UINT64), p += sizeof(UINT64)) {
  0081b  8b 5d 0c  mov      ebx, DWORD PTR _wrdlen$[ebp]
  0081e  56                push     esi
  0081f  57                push     edi
  00820  ba c5 9d 1c 81    mov      edx, -2128831035 ; 811c9dc5H
  00825  83 fb 08  cmp      ebx, 8
  00828  72 28             jb       SHORT $LN5@FNV1A_Hash@2
  0082a  8b fb             mov      edi, ebx
  0082c  c1 ef 03  shr      edi, 3
  0082f  90                npad     1
$LL7@FNV1A_Hash@2:
; 779  :          // Variant #1:
; 780  :          //hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
; 781  :          //hash32 = (hash32 ^ *(DWORD *)(p+4)) * PRIME;
; 782  :          // Variant #2:
; 783  :          OneAccess = *(UINT64 *)p;
  00830  8b 01             mov      eax, DWORD PTR [ecx]
  00832  8b 71 04  mov      esi, DWORD PTR [ecx+4]
; 784  :          hash32 = (hash32 ^ ((OneAccess>>0)&0xFFFFFFFF)) * PRIME;
  00835  33 c2             xor      eax, edx
  00837  69 c0 e7 d3 0a
         00                imul     eax, 709607              ; 000ad3e7H
; 785  :          hash32 = (hash32 ^ ((OneAccess>>32)&0xFFFFFFFF)) * PRIME;  // No need for ANDing but compiler is smart enough to see
that.
  0083d  33 c6             xor      eax, esi
  0083f  69 c0 e7 d3 0a
         00                imul     eax, 709607              ; 000ad3e7H
  00845  83 eb 08  sub      ebx, 8
  00848  83 c1 08  add      ecx, 8
  0084b  83 ef 01  sub      edi, 1
  0084e  8b d0             mov      edx, eax
  00850  75 de             jne      SHORT $LL7@FNV1A_Hash@2
$LN5@FNV1A_Hash@2:
; 786  : }
; 787  : for(; wrdlen >= sizeof(WORD); wrdlen -= sizeof(WORD), p += sizeof(WORD)) {
  00852  83 fb 02  cmp      ebx, 2
  00855  72 21             jb       SHORT $LN2@FNV1A_Hash@2
  00857  8b c3             mov      eax, ebx
  00859  d1 e8             shr      eax, 1
  0085b  eb 03 8d 49 00    npad     5
$LL4@FNV1A_Hash@2:
; 788  :          hash32 = (hash32 ^ *(WORD*)p) * PRIME;
  00860  0f b7 31  movzx    esi, WORD PTR [ecx]
  00863  33 f2             xor      esi, edx
  00865  69 f6 e7 d3 0a
         00                imul     esi, 709607              ; 000ad3e7H
  0086b  83 eb 02  sub      ebx, 2
  0086e  83 c1 02  add      ecx, 2
  00871  83 e8 01  sub      eax, 1
  00874  8b d6             mov      edx, esi
  00876  75 e8             jne      SHORT $LL4@FNV1A_Hash@2
$LN2@FNV1A_Hash@2:
; 789  : }
; 790  : if (wrdlen & 1)
  00878  f6 c3 01  test     bl, 1
  0087b  74 0d             je       SHORT $LN1@FNV1A_Hash@2
; 791  :          hash32 = (hash32 ^ *p) * PRIME;
```

```
  0087d  0f be 01  movsx    eax, BYTE PTR [ecx]
  00880  33 c2              xor      eax, edx
  00882  69 c0 e7 d3 0a
         00                 imul     eax, 709607              ; 000ad3e7H
  00888  8b d0              mov      edx, eax
$LN1@FNV1A_Hash@2:
; 792  :
; 793  : return hash32 ^ (hash32 >> 16);
; 794  : }
  0088a  5f                 pop      edi
  0088b  8b c2              mov      eax, edx
  0088d  c1 e8 10  shr      eax, 16                  ; 00000010H
  00890  5e                 pop      esi
  00891  33 c2              xor      eax, edx
  00893  5b                 pop      ebx
  00894  8b e5              mov      esp, ebp
  00896  5d                 pop      ebp
  00897  c3                 ret      0
?FNV1A_Hash_Peregrine@@YAIPBDK@Z ENDP                      ; FNV1A_Hash_Peregrine
?FNV1A_Hash_Smaragd@@YAIPBDK@Z PROC                  ; FNV1A_Hash_Smaragd
; 735  : const UINT PRIME = 709607;//15607;
; 736  :
; 737  : //Smaragd 1607: [2088443]
; 738  : //Smaragd 15607: [2093321]
; 739  : //Smaragd 16607: [2084267]
; 740  : //Smaragd 709607: [2083208]
; 741  :
; 742  : UINT hash32 = 2166136261;
; 743  : const char *p = str;
  008a0  8b 54 24 04        mov      edx, DWORD PTR _str$[esp-4]
  008a4  53                 push     ebx
; 744  : UINT OneAccess;
; 745  :
; 746  : for(; wrdlen >= sizeof(UINT); wrdlen -= sizeof(UINT), p += sizeof(UINT)) {
  008a5  8b 5c 24 0c        mov      ebx, DWORD PTR _wrdlen$[esp]
  008a9  b9 c5 9d 1c 81     mov      ecx, -2128831035 ; 811c9dc5H
  008ae  83 fb 04  cmp      ebx, 4
  008b1  72 34              jb       SHORT $LN3@FNV1A_Hash@3
  008b3  56                 push     esi
  008b4  8b f3              mov      esi, ebx
  008b6  57                 push     edi
  008b7  c1 ee 02  shr      esi, 2
  008ba  8d 9b 00 00 00
         00                 npad     6
$LL5@FNV1A_Hash@3:
; 747  :          // Variant #1:
; 748  :          //hash32 = (hash32 ^ *(WORD*)(p+0)) * PRIME;
; 749  :          //hash32 = (hash32 ^ *(WORD*)(p+2)) * PRIME;
; 750  :          // Variant #2:
; 751  :          OneAccess = *(UINT *)p;
  008c0  8b 02              mov      eax, DWORD PTR [edx]
; 752  :          hash32 = (hash32 ^ ((OneAccess>>0)&0xFFFF)) * PRIME;
  008c2  0f b7 f8  movzx    edi, ax
  008c5  33 f9              xor      edi, ecx
  008c7  69 ff e7 d3 0a
         00                 imul     edi, 709607              ; 000ad3e7H
; 753  :          hash32 = (hash32 ^ ((OneAccess>>16)&0xFFFF)) * PRIME;  // No need for ANDing but compiler is smart enough to see that.
  008cd  c1 e8 10  shr      eax, 16                  ; 00000010H
  008d0  33 f8              xor      edi, eax
  008d2  69 ff e7 d3 0a
         00                 imul     edi, 709607              ; 000ad3e7H
  008d8  83 eb 04  sub      ebx, 4
```

```
  008db  83 c2 04  add      edx, 4
  008de  83 ee 01  sub      esi, 1
  008e1  8b cf              mov      ecx, edi
  008e3  75 db              jne      SHORT $LL5@FNV1A_Hash@3
  008e5  5f                 pop      edi
  008e6  5e                 pop      esi
$LN3@FNV1A_Hash@3:
; 754  : }
; 755  : if (wrdlen & sizeof(WORD)) {
  008e7  f6 c3 02  test     bl, 2
  008ea  74 10              je       SHORT $LN2@FNV1A_Hash@3
; 756  :          hash32 = (hash32 ^ *(WORD*)p) * PRIME;
  008ec  0f b7 02  movzx    eax, WORD PTR [edx]
  008ef  33 c1              xor      eax, ecx
  008f1  69 c0 e7 d3 0a
         00                 imul     eax, 709607              ; 000ad3e7H
  008f7  8b c8              mov      ecx, eax
; 757  :          p += sizeof(WORD);
  008f9  83 c2 02  add      edx, 2
$LN2@FNV1A_Hash@3:
; 758  : }
; 759  : if (wrdlen & 1)
  008fc  f6 c3 01  test     bl, 1
  008ff  5b                 pop      ebx
  00900  74 0d              je       SHORT $LN1@FNV1A_Hash@3
; 760  :          hash32 = (hash32 ^ *p) * PRIME;
  00902  0f be 12  movsx    edx, BYTE PTR [edx]
  00905  33 d1              xor      edx, ecx
  00907  69 d2 e7 d3 0a
         00                 imul     edx, 709607              ; 000ad3e7H
  0090d  8b ca              mov      ecx, edx
$LN1@FNV1A_Hash@3:
; 761  :
; 762  : return hash32 ^ (hash32 >> 16);
  0090f  8b c1              mov      eax, ecx
  00911  c1 e8 10  shr      eax, 16                  ; 00000010H
  00914  33 c1              xor      eax, ecx
; 763  : }
  00916  c3                 ret      0
?FNV1A_Hash_Smaragd@@YAIPBDK@Z ENDP                  ; FNV1A_Hash_Smaragd
?Hash_Sixtinsensitive_Boosted@@YAIPBDK@Z PROC        ; Hash_Sixtinsensitive_Boosted
; 1027 : UINT hash = 2166136261;
; 1028 : UINT hashBUFFER_EAX, hashBUFFER_EDX;
; 1029 : const CHAR * p = str;
; 1030 :
; 1031 : // 0x41 = 065 'A' 010 [0 0001]
; 1032 : // 0x5A = 090 'Z' 010 [1 1010]
; 1033 : // 0x61 = 097 'a' 011 [0 0001]
; 1034 : // 0x7A = 122 'z' 011 [1 1010]
; 1035 :
; 1036 : // Reduce the number of multiplications by unrolling the loop
; 1037 : for(; wrdlen >= 12; wrdlen -= 12, p += 12) {
  002c0  8b 54 24 08        mov      edx, DWORD PTR _wrdlen$[esp-4]
  002c4  8b 4c 24 04        mov      ecx, DWORD PTR _str$[esp-4]
  002c8  53                 push     ebx
  002c9  56                 push     esi
  002ca  57                 push     edi
  002cb  bb c5 9d 1c 81     mov      ebx, -2128831035 ; 811c9dc5H
  002d0  83 fa 0c  cmp      edx, 12                  ; 0000000cH
  002d3  0f 82 ae 00 00
         00                 jb       $LN6@Hash_Sixti
  002d9  b8 ab aa aa aa     mov      eax, -1431655765 ; aaaaaaabH
```

```
  002de  f7 e2              mul      edx
  002e0  c1 ea 03  shr      edx, 3
  002e3  89 54 24 10        mov      DWORD PTR tv165[esp+8], edx
  002e7  55                 push     ebp
  002e8  eb 06 8d 9b 00
         00 00 00  npad     8
$LL8@Hash_Sixti:
; 1038 :          //hashBUFFER_AX = (*(DWORD*)(p+0)&0xFFFF);
; 1039 :          hashBUFFER_EAX = (*(DWORD*)(p+0)&0x1F1F1F1F);
; 1040 :          hashBUFFER_EDX = (*(DWORD*)(p+4)&0x1F1F1F1F);
; 1041 :          //hashBUFFER_EBX = (*(DWORD*)(p+8)&0x1F1F1F1F);
; 1042 :
; 1043 :          //hashBUFFER_BL = (*(p+8)&0x1F);
; 1044 :          //hashBUFFER_BH = (*(p+9)&0x1F);
; 1045 :          //hashBUFFER_BL2 = (*(p+10)&0x1F);
; 1046 :          //hashBUFFER_BH2 = (*(p+11)&0x1F);
; 1047 :
; 1048 :          //12bytes-in-8bytes or 96bits-to-60bits
; 1049 :          // Two times next:
; 1050 :          //6bytes-in-4bytes or 48bits-to-30bits
; 1051 :          // Two times next:
; 1052 :          //3bytes-in-2bytes or 24bits-to-15bits
; 1053 :          //EAX                    BL          BH
; 1054 :          //[5bit][3bit][5bit][3bit][5bit][3bit][5bit][3bit]
; 1055 :          //       5th[0..15]  13th[0..15]
; 1056 :          //       BL lower 3  BL higher 2bits
; 1057 :          // OR or XOR no difference
; 1058 :          hashBUFFER_EAX = hashBUFFER_EAX ^ (((*(p+8)&0x1F)&0x07)<<5); // BL lower 3bits of 5bits
; 1059 :          hashBUFFER_EAX = hashBUFFER_EAX ^ (((*(p+8)&0x1F)&0x18)<<(2+8)); // BL higher 2bits of 5bits
; 1060 :          hashBUFFER_EAX = hashBUFFER_EAX ^ (((*(p+9)&0x1F)&0x07)<<(5+16)); // BH lower 3bits of 5bits
; 1061 :          hashBUFFER_EAX = hashBUFFER_EAX ^ (((*(p+9)&0x1F)&0x18)<<((2+8)+16)); // BH higher 2bits of 5bits
; 1062 :          hash = ((hash ^ hashBUFFER_EAX)<<5) - (hash ^ hashBUFFER_EAX);
; 1063 :          hashBUFFER_EDX = hashBUFFER_EDX ^ (((*(p+10)&0x1F)&0x07)<<5); // BL lower 3bits of 5bits
; 1064 :          hashBUFFER_EDX = hashBUFFER_EDX ^ (((*(p+10)&0x1F)&0x18)<<(2+8)); // BL higher 2bits of 5bits
; 1065 :          hashBUFFER_EDX = hashBUFFER_EDX ^ (((*(p+11)&0x1F)&0x07)<<(5+16)); // BH lower 3bits of 5bits
; 1066 :          hashBUFFER_EDX = hashBUFFER_EDX ^ (((*(p+11)&0x1F)&0x18)<<((2+8)+16)); // BH higher 2bits of 5bits
; 1067 :          hash = ((hash ^ hashBUFFER_EDX)<<5) - (hash ^ hashBUFFER_EDX);
  002f0  0f be 71 09        movsx    esi, BYTE PTR [ecx+9]
  002f4  0f be 79 08        movsx    edi, BYTE PTR [ecx+8]
  002f8  8b c6              mov      eax, esi
  002fa  83 e6 07  and      esi, 7
  002fd  83 e0 18  and      eax, 24                 ; 00000018H
  00300  0f be 69 0b        movsx    ebp, BYTE PTR [ecx+11]
  00304  c1 e0 05  shl      eax, 5
  00307  33 c6              xor      eax, esi
  00309  0f be 51 0a        movsx    edx, BYTE PTR [ecx+10]
  0030d  c1 e0 0b  shl      eax, 11                 ; 0000000bH
  00310  8b f7              mov      esi, edi
  00312  83 e6 18  and      esi, 24                 ; 00000018H
  00315  33 c6              xor      eax, esi
  00317  8b 31              mov      esi, DWORD PTR [ecx]
  00319  81 e6 1f 1f 1f
         1f                 and      esi, 522133279          ; 1f1f1f1fH
  0031f  c1 e0 05  shl      eax, 5
  00322  83 e7 07  and      edi, 7
  00325  33 c7              xor      eax, edi
  00327  c1 e0 05  shl      eax, 5
  0032a  33 c6              xor      eax, esi
  0032c  33 c3              xor      eax, ebx
  0032e  8b f0              mov      esi, eax
  00330  c1 e6 05  shl      esi, 5
  00333  2b f0              sub      esi, eax
```

```
  00335  8b c6              mov      eax, esi
  00337  83 6c 24 18 0c     sub      DWORD PTR _wrdlen$[esp+12], 12 ; 0000000cH
  0033c  8b f5              mov      esi, ebp
  0033e  83 e6 18  and      esi, 24                 ; 00000018H
  00341  c1 e6 05  shl      esi, 5
  00344  8b fa              mov      edi, edx
  00346  83 e5 07  and      ebp, 7
  00349  33 f5              xor      esi, ebp
  0034b  c1 e6 0b  shl      esi, 11                 ; 0000000bH
  0034e  83 e2 07  and      edx, 7
  00351  83 e7 18  and      edi, 24                 ; 00000018H
  00354  33 f7              xor      esi, edi
  00356  c1 e6 05  shl      esi, 5
  00359  33 f2              xor      esi, edx
  0035b  8b 51 04  mov      edx, DWORD PTR [ecx+4]
  0035e  81 e2 1f 1f 1f
         1f                 and      edx, 522133279          ; 1f1f1f1fH
  00364  c1 e6 05  shl      esi, 5
  00367  33 c6              xor      eax, esi
  00369  33 c2              xor      eax, edx
  0036b  8b d0              mov      edx, eax
  0036d  c1 e2 05  shl      edx, 5
  00370  2b d0              sub      edx, eax
  00372  83 c1 0c  add      ecx, 12                 ; 0000000cH
  00375  83 6c 24 14 01     sub      DWORD PTR tv165[esp+12], 1
  0037a  8b da              mov      ebx, edx
  0037c  0f 85 6e ff ff
         ff                 jne      $LL8@Hash_Sixti
  00382  8b 54 24 18        mov      edx, DWORD PTR _wrdlen$[esp+12]
  00386  5d                 pop      ebp
$LN6@Hash_Sixti:
; 1068 :     //31:   12170359  11983592  11991947  11997376  12005131|  11983592 [2251734]
; 1069 : }
; 1070 : /*
; 1071 : // Post-Variant #1:
; 1072 : for(; wrdlen; wrdlen--, p++) {
; 1073 :          hash = ((hash ^ (*p&0x1F))<<5) - (hash ^ (*p&0x1F));
; 1074 : }
; 1075 : */
; 1076 : // Post-Variant #2:
; 1077 : for(; wrdlen >= sizeof(DWORD); wrdlen -= sizeof(DWORD), p += sizeof(DWORD)) {
  00387  83 fa 04  cmp      edx, 4
  0038a  72 2a              jb       SHORT $LN3@Hash_Sixti
  0038c  8b f2              mov      esi, edx
  0038e  c1 ee 02  shr      esi, 2
$LL5@Hash_Sixti:
; 1078 :          hash = ((hash ^ (*(DWORD*)(p)&0x1F1F1F1F))<<5) - (hash ^ (*(DWORD*)(p)&0x1F1F1F1F));
  00391  8b 01              mov      eax, DWORD PTR [ecx]
  00393  8b f8              mov      edi, eax
  00395  81 e7 1f 1f 1f
         ff                 and      edi, -14737633          ; ff1f1f1fH
  0039b  33 fb              xor      edi, ebx
  0039d  25 1f 1f 1f 1f     and      eax, 522133279          ; 1f1f1f1fH
  003a2  33 c3              xor      eax, ebx
  003a4  c1 e7 05  shl      edi, 5
  003a7  2b f8              sub      edi, eax
  003a9  83 ea 04  sub      edx, 4
  003ac  83 c1 04  add      ecx, 4
  003af  83 ee 01  sub      esi, 1
  003b2  8b df              mov      ebx, edi
  003b4  75 db              jne      SHORT $LL5@Hash_Sixti
$LN3@Hash_Sixti:
```

```
; 1079 : }
; 1080 : if (wrdlen & sizeof(WORD)) {
  003b6  f6 c2 02  test     dl, 2
  003b9  74 16              je       SHORT $LN2@Hash_Sixti
; 1081 :          hash = ((hash ^ (*(WORD*)p&0x1F1F))<<5) - (hash ^ (*(WORD*)p&0x1F1F));
  003bb  0f b7 01  movzx    eax, WORD PTR [ecx]
  003be  25 1f 1f 00 00     and      eax, 7967                ; 00001f1fH
  003c3  33 c3              xor      eax, ebx
  003c5  8b f0              mov      esi, eax
  003c7  c1 e6 05  shl      esi, 5
  003ca  2b f0              sub      esi, eax
  003cc  8b de              mov      ebx, esi
; 1082 :          p += sizeof(WORD);
  003ce  83 c1 02  add      ecx, 2
$LN2@Hash_Sixti:
; 1083 : }
; 1084 : if (wrdlen & 1)
  003d1  f6 c2 01  test     dl, 1
  003d4  74 11              je       SHORT $LN1@Hash_Sixti
; 1085 :          hash = ((hash ^ (*p&0x1F))<<5) - (hash ^ (*p&0x1F));
  003d6  0f be 01  movsx    eax, BYTE PTR [ecx]
  003d9  83 e0 1f  and      eax, 31                  ; 0000001fH
  003dc  33 c3              xor      eax, ebx
  003de  8b c8              mov      ecx, eax
  003e0  c1 e1 05  shl      ecx, 5
  003e3  2b c8              sub      ecx, eax
  003e5  8b d9              mov      ebx, ecx
$LN1@Hash_Sixti:
; 1086 :
; 1087 : return (hash>>16) ^ hash;
  003e7  8b c3              mov      eax, ebx
  003e9  5f                 pop      edi
  003ea  c1 e8 10  shr      eax, 16                  ; 00000010H
  003ed  5e                 pop      esi
  003ee  33 c3              xor      eax, ebx
  003f0  5b                 pop      ebx
; 1088 : }
  003f1  c3                 ret      0
?Hash_Sixtinsensitive_Boosted@@YAIPBDK@Z ENDP         ; Hash_Sixtinsensitive_Boosted
?Hash_Sixtinsensitive@@YAIPBDK@Z PROC                         ; Hash_Sixtinsensitive
; 958  : {
  00400  53                 push     ebx
  00401  55                 push     ebp
; 959  : UINT hash = 2166136261;
; 960  : UINT hashBUFFER_EAX, hashBUFFER_BH, hashBUFFER_BL;
; 961  : const CHAR * p = str;
; 962  :
; 963  : // 0x41 = 065 'A' 010 [0 0001]
; 964  : // 0x5A = 090 'Z' 010 [1 1010]
; 965  : // 0x61 = 097 'a' 011 [0 0001]
; 966  : // 0x7A = 122 'z' 011 [1 1010]
; 967  :
; 968  : // Reduce the number of multiplications by unrolling the loop
; 969  : for(; wrdlen >= 6; wrdlen -= 6, p += 6) {
  00402  8b 6c 24 10        mov      ebp, DWORD PTR _wrdlen$[esp+4]
  00406  57                 push     edi
  00407  8b 7c 24 10        mov      edi, DWORD PTR _str$[esp+8]
  0040b  bb c5 9d 1c 81     mov      ebx, -2128831035 ; 811c9dc5H
  00410  83 fd 06  cmp      ebp, 6
  00413  72 5c              jb       SHORT $LN4@Hash_Sixti@2
  00415  b8 ab aa aa aa     mov      eax, -1431655765 ; aaaaaaabH
  0041a  f7 e5              mul      ebp
```

```
  0041c  c1 ea 02  shr      edx, 2
  0041f  56                 push     esi
$LL6@Hash_Sixti@2:
; 970  :          //hashBUFFER_AX = (*(DWORD*)(p+0)&0xFFFF);
; 971  :          hashBUFFER_EAX = (*(DWORD*)(p+0)&0x1F1F1F1F);
; 972  :          hashBUFFER_BL = (*(p+4)&0x1F);
; 973  :          hashBUFFER_BH = (*(p+5)&0x1F);
  00420  0f be 77 05        movsx    esi, BYTE PTR [edi+5]
  00424  0f be 4f 04        movsx    ecx, BYTE PTR [edi+4]
  00428  83 e6 1f  and      esi, 31                  ; 0000001fH
; 974  :          //6bytes-in-4bytes or 48bits-to-30bits
; 975  :          // Two times next:
; 976  :          //3bytes-in-2bytes or 24bits-to-15bits
; 977  :          //EAX                    BL          BH
; 978  :          //[5bit][3bit][5bit][3bit][5bit][3bit][5bit][3bit]
; 979  :          //       5th[0..15]  13th[0..15]
; 980  :          //       BL lower 3  BL higher 2bits
; 981  :          // OR or XOR no difference
; 982  :          hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BL&0x07)<<5); // BL lower 3bits of 5bits
; 983  :          hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BL&0x18)<<(2+8)); // BL higher 2bits of 5bits
; 984  :          hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BH&0x07)<<(5+16)); // BH lower 3bits of 5bits
; 985  :          hashBUFFER_EAX = hashBUFFER_EAX ^ ((hashBUFFER_BH&0x18)<<((2+8)+16)); // BH higher 2bits of 5bits
; 986  :          //hash = (hash ^ hashBUFFER_EAX)*1607; //What a mess: <<7 becomes imul but <<5 not!?
; 987  :          hash = ((hash ^ hashBUFFER_EAX)<<5) - (hash ^ hashBUFFER_EAX);
  0042b  8b c6              mov      eax, esi
  0042d  83 e0 18  and      eax, 24                  ; 00000018H
  00430  c1 e0 05  shl      eax, 5
  00433  83 e1 1f  and      ecx, 31                  ; 0000001fH
  00436  83 e6 07  and      esi, 7
  00439  33 c6              xor      eax, esi
  0043b  8b f1              mov      esi, ecx
  0043d  c1 e0 0b  shl      eax, 11                  ; 0000000bH
  00440  83 e1 07  and      ecx, 7
  00443  83 e6 18  and      esi, 24                  ; 00000018H
  00446  33 c6              xor      eax, esi
  00448  c1 e0 05  shl      eax, 5
  0044b  33 c1              xor      eax, ecx
  0044d  8b 0f              mov      ecx, DWORD PTR [edi]
  0044f  81 e1 1f 1f 1f
         1f                 and      ecx, 522133279           ; 1f1f1f1fH
  00455  c1 e0 05  shl      eax, 5
  00458  33 c1              xor      eax, ecx
  0045a  33 c3              xor      eax, ebx
  0045c  8b c8              mov      ecx, eax
  0045e  c1 e1 05  shl      ecx, 5
  00461  2b c8              sub      ecx, eax
  00463  83 ed 06  sub      ebp, 6
  00466  83 c7 06  add      edi, 6
  00469  83 ea 01  sub      edx, 1
  0046c  8b d9              mov      ebx, ecx
  0046e  75 b0              jne      SHORT $LL6@Hash_Sixti@2
  00470  5e                 pop      esi
$LN4@Hash_Sixti@2:
; 988  :          //1607:[2118599]
; 989  :          // 127:[2121081]
; 990  :          //  31:[2139242]
; 991  :          //  17:[2150803]
; 992  :          //   7:[2166336]
; 993  :          //   5:[2183044]
; 994  :          //8191:[2200477]
; 995  :          //   3:[2205095]
; 996  :          // 257:[2206188]
```

```
; 997  : }
; 998  : // Post-Variant #1:
; 999  : for(; wrdlen; wrdlen--, p++) {
  00471  85 ed              test     ebp, ebp
  00473  74 17              je       SHORT $LN1@Hash_Sixti@2
$LL3@Hash_Sixti@2:
; 1000 :          hash = ((hash ^ (*p&0x1F))<<5) - (hash ^ (*p&0x1F));
  00475  0f be 07  movsx    eax, BYTE PTR [edi]
  00478  83 e0 1f  and      eax, 31                 ; 0000001fH
  0047b  33 c3              xor      eax, ebx
  0047d  8b d0              mov      edx, eax
  0047f  c1 e2 05  shl      edx, 5
  00482  2b d0              sub      edx, eax
  00484  4d                 dec      ebp
  00485  47                 inc      edi
  00486  8b da              mov      ebx, edx
  00488  85 ed              test     ebp, ebp
  0048a  75 e9              jne      SHORT $LL3@Hash_Sixti@2
$LN1@Hash_Sixti@2:
; 1001 : }
; 1002 : /*
; 1003 : // Post-Variant #2:
; 1004 : for(; wrdlen >= 2; wrdlen -= 2, p += 2) {
; 1005 : hash = ((hash ^ (*(DWORD*)p&0xFFFF))<<5) - (hash ^ (*(DWORD*)p&0xFFFF));
; 1006 : }
; 1007 : if (wrdlen & 1)
; 1008 : hash = ((hash ^ *p)<<5) - (hash ^ *p);
; 1009 : */
; 1010 : /*
; 1011 : // Post-Variant #3:
; 1012 : for(; wrdlen >= 4; wrdlen -= 4, p += 4) {
; 1013 : hash = ((hash ^ *(DWORD*)p)<<5) - (hash ^ *(DWORD*)p);
; 1014 : }
; 1015 : if (wrdlen & 2) {
; 1016 : hash = ((hash ^ (*(WORD*)p))<<5) - (hash ^ (*(WORD*)p));
; 1017 : p++;p++;
; 1018 : }
; 1019 : if (wrdlen & 1)
; 1020 : hash = ((hash ^ *p)<<5) - (hash ^ *p);
; 1021 : */
; 1022 : return (hash>>16) ^ hash;
  0048c  8b c3              mov      eax, ebx
  0048e  5f                 pop      edi
  0048f  c1 e8 10  shr      eax, 16                 ; 00000010H
  00492  5d                 pop      ebp
  00493  33 c3              xor      eax, ebx
  00495  5b                 pop      ebx
; 1023 : }
  00496  c3                 ret      0
?Hash_Sixtinsensitive@@YAIPBDK@Z ENDP                       ; Hash_Sixtinsensitive
?HashAlfalfa_QWORD@@YAIPBDK@Z PROC                  ; HashAlfalfa_QWORD
; 843  : {
  00580  55                 push     ebp
  00581  8b ec              mov      ebp, esp
  00583  83 e4 f8  and      esp, -8                 ; fffffff8H
  00586  83 ec 14  sub      esp, 20                 ; 00000014H
; 844  : UINT64 hashQWORD;
; 845  : UINT hashAlfalfa = 7;
; 846  :
; 847  : for(; wrdlen >= 8; wrdlen -= 8, key += 8) {
  00589  8b 45 0c  mov      eax, DWORD PTR _wrdlen$[ebp]
  0058c  8b 55 08  mov      edx, DWORD PTR _key$[ebp]
```

```
  0058f  53                push     ebx
  00590  56                push     esi
  00591  b9 07 00 00 00     mov      ecx, 7
  00596  57                push     edi
  00597  89 4c 24 10        mov      DWORD PTR _hashAlfalfa$[esp+32], ecx
  0059b  83 f8 08  cmp      eax, 8
  0059e  0f 82 be 00 00
         00                jb       $LN4@HashAlfalf
  005a4  c1 e8 03  shr      eax, 3
  005a7  89 44 24 14        mov      DWORD PTR tv184[esp+32], eax
  005ab  eb 03 8d 49 00     npad     5
$LL6@HashAlfalf:
; 848  :          hashQWORD=*(unsigned long long*)key;
  005b0  8b 42 04  mov      eax, DWORD PTR [edx+4]
  005b3  8b 0a              mov      ecx, DWORD PTR [edx]
; 849  :          hashAlfalfa = (53) * ((53) * hashAlfalfa + ((hashQWORD>>0)&0xFF) ) + ((hashQWORD>>8)&0xFF);
; 850  :          hashAlfalfa = (53) * ((53) * hashAlfalfa + ((hashQWORD>>16)&0xFF) ) + ((hashQWORD>>24)&0xFF);
  005b5  8b f8              mov      edi, eax
  005b7  8b f1              mov      esi, ecx
  005b9  0f ac fe 10        shrd     esi, edi, 16
  005bd  81 e6 ff 00 00
         00                and      esi, 255         ; 000000ffH
  005c3  c1 ef 10  shr      edi, 16                   ; 00000010H
; 851  :          hashAlfalfa = (53) * ((53) * hashAlfalfa + ((hashQWORD>>32)&0xFF) ) + ((hashQWORD>>40)&0xFF);
; 852  :          hashAlfalfa = (53) * ((53) * hashAlfalfa + ((hashQWORD>>48)&0xFF) ) + ((hashQWORD>>56)&0xFF);
  005c6  6b f6 35  imul     esi, 53                   ; 00000035H
  005c9  8b d8              mov      ebx, eax
  005cb  8b f9              mov      edi, ecx
  005cd  0f ac df 18        shrd     edi, ebx, 24
  005d1  81 e7 ff 00 00
         00                and      edi, 255         ; 000000ffH
  005d7  03 f7              add      esi, edi
  005d9  6b f6 35  imul     esi, 53                   ; 00000035H
  005dc  0f b6 f8  movzx    edi, al
  005df  03 f7              add      esi, edi
  005e1  6b f6 35  imul     esi, 53                   ; 00000035H
  005e4  8b f8              mov      edi, eax
  005e6  c1 ef 08  shr      edi, 8
  005e9  81 e7 ff 00 00
         00                and      edi, 255         ; 000000ffH
  005ef  03 f7              add      esi, edi
  005f1  6b f6 35  imul     esi, 53                   ; 00000035H
  005f4  8b f8              mov      edi, eax
  005f6  c1 ef 10  shr      edi, 16                   ; 00000010H
  005f9  81 e7 ff 00 00
         00                and      edi, 255         ; 000000ffH
  005ff  03 f7              add      esi, edi
  00601  c1 eb 18  shr      ebx, 24                   ; 00000018H
  00604  6b f6 35  imul     esi, 53                   ; 00000035H
  00607  8b d8              mov      ebx, eax
  00609  8b f9              mov      edi, ecx
  0060b  0f ac df 08        shrd     edi, ebx, 8
  0060f  81 e7 ff 00 00
         00                and      edi, 255         ; 000000ffH
  00615  69 ff a9 4f 19
         29                imul     edi, 689524649           ; 29194fa9H
  0061b  89 44 24 1c        mov      DWORD PTR _hashQWORD$[esp+36], eax
  0061f  c1 e8 18  shr      eax, 24                   ; 00000018H
  00622  25 ff 00 00 00     and      eax, 255         ; 000000ffH
  00627  03 f7              add      esi, edi
  00629  03 f0              add      esi, eax
  0062b  0f b6 c1  movzx    eax, cl
```

```
  0062e  8b 4c 24 10        mov      ecx, DWORD PTR _hashAlfalfa$[esp+32]
  00632  69 c0 03 82 c2
         7d                 imul     eax, 2109899267           ; 7dc28203H
  00638  69 c9 9f ea 44
         09                 imul     ecx, 155511455            ; 0944ea9fH
  0063e  2b f0              sub      esi, eax
  00640  8b 45 0c  mov      eax, DWORD PTR _wrdlen$[ebp]
  00643  2b f1              sub      esi, ecx
  00645  83 e8 08  sub      eax, 8
  00648  c1 eb 08  shr      ebx, 8
  0064b  83 c2 08  add      edx, 8
  0064e  83 6c 24 14 01     sub      DWORD PTR tv184[esp+32], 1
  00653  8b ce              mov      ecx, esi
  00655  89 74 24 10        mov      DWORD PTR _hashAlfalfa$[esp+32], esi
  00659  89 45 0c  mov      DWORD PTR _wrdlen$[ebp], eax
  0065c  0f 85 4e ff ff
         ff                 jne      $LL6@HashAlfalf
$LN4@HashAlfalf:
; 853  : }
; 854  : for(; wrdlen; wrdlen--, key++)
  00662  85 c0              test     eax, eax
  00664  74 0e              je       SHORT $LN1@HashAlfalf
$LL3@HashAlfalf:
; 855  :          hashAlfalfa = (53) * hashAlfalfa + *key;
  00666  0f be 32  movsx    esi, BYTE PTR [edx]
  00669  6b c9 35  imul     ecx, 53                   ; 00000035H
  0066c  48                 dec      eax
  0066d  03 ce              add      ecx, esi
  0066f  42                 inc      edx
  00670  85 c0              test     eax, eax
  00672  75 f2              jne      SHORT $LL3@HashAlfalf
$LN1@HashAlfalf:
; 856  : return hashAlfalfa ^ (hashAlfalfa >> 16);
; 857  : }
  00674  5f                 pop      edi
  00675  8b c1              mov      eax, ecx
  00677  c1 e8 10  shr      eax, 16                   ; 00000010H
  0067a  5e                 pop      esi
  0067b  33 c1              xor      eax, ecx
  0067d  5b                 pop      ebx
  0067e  8b e5              mov      esp, ebp
  00680  5d                 pop      ebp
  00681  c3                 ret      0
?HashAlfalfa_QWORD@@YAIPBDK@Z ENDP                    ; HashAlfalfa_QWORD
?HashAlfalfa_DWORD@@YAIPBDK@Z PROC                    ; HashAlfalfa_DWORD
; 828  : {
  00690  53                 push     ebx
; 829  : SIZE_T i, j;
; 830  : UINT hash = 7;
; 831  : for(i = 0; i < (wrdlen & -(INT_PTR)sizeof(DWORD)); i += sizeof(DWORD)) {
  00691  8b 5c 24 0c        mov      ebx, DWORD PTR _wrdlen$[esp]
  00695  56                 push     esi
  00696  8b f3              mov      esi, ebx
  00698  33 d2              xor      edx, edx
  0069a  83 e6 fc  and      esi, -4                   ; fffffffcH
  0069d  57                 push     edi
  0069e  8b 7c 24 10        mov      edi, DWORD PTR _key$[esp+8]
  006a2  b9 07 00 00 00     mov      ecx, 7
  006a7  76 44              jbe      SHORT $LN4@HashAlfalf@2
  006a9  55                 push     ebp
  006aa  8d 9b 00 00 00
         00                 npad     6
```

```
$LL6@HashAlfalf@2:
; 832  :          DWORD x = *(DWORD*)(key + i);
  006b0  8b 04 3a  mov      eax, DWORD PTR [edx+edi]
; 833  :          hash = (53) * ((53) * hash + ((x>>0)&0xFF) ) + ((x>>8)&0xFF);
; 834  :          hash = (53) * ((53) * hash + ((x>>16)&0xFF) ) + ((x>>24)&0xFF);
  006b3  6b c9 35  imul     ecx, 53                 ; 00000035H
  006b6  0f b6 e8  movzx    ebp, al
  006b9  03 e9              add      ebp, ecx
  006bb  6b ed 35  imul     ebp, 53                 ; 00000035H
  006be  8b c8              mov      ecx, eax
  006c0  c1 e9 08  shr      ecx, 8
  006c3  81 e1 ff 00 00
         00                 and      ecx, 255        ; 000000ffH
  006c9  03 e9              add      ebp, ecx
  006cb  6b ed 35  imul     ebp, 53                 ; 00000035H
  006ce  8b c8              mov      ecx, eax
  006d0  c1 e9 10  shr      ecx, 16                 ; 00000010H
  006d3  81 e1 ff 00 00
         00                 and      ecx, 255        ; 000000ffH
  006d9  03 e9              add      ebp, ecx
  006db  6b ed 35  imul     ebp, 53                 ; 00000035H
  006de  c1 e8 18  shr      eax, 24                 ; 00000018H
  006e1  03 e8              add      ebp, eax
  006e3  83 c2 04  add      edx, 4
  006e6  8b cd              mov      ecx, ebp
  006e8  3b d6              cmp      edx, esi
  006ea  72 c4              jb       SHORT $LL6@HashAlfalf@2
  006ec  5d                 pop      ebp
$LN4@HashAlfalf@2:
; 835  : }
; 836  : for(j = 0; j < (wrdlen & (sizeof(DWORD) - 1)); j++) {
  006ed  33 c0              xor      eax, eax
  006ef  83 e3 03  and      ebx, 3
  006f2  8b f3              mov      esi, ebx
  006f4  76 10              jbe      SHORT $LN1@HashAlfalf@2
  006f6  03 d7              add      edx, edi
$LL3@HashAlfalf@2:
; 837  :          hash = (53) * hash + key[i+j];
  006f8  0f be 3c 02        movsx    edi, BYTE PTR [edx+eax]
  006fc  6b c9 35  imul     ecx, 53                 ; 00000035H
  006ff  40                 inc      eax
  00700  03 cf              add      ecx, edi
  00702  3b c6              cmp      eax, esi
  00704  72 f2              jb       SHORT $LL3@HashAlfalf@2
$LN1@HashAlfalf@2:
  00706  5f                 pop      edi
; 838  : }
; 839  : return hash ^ (hash >> 16);
  00707  8b c1              mov      eax, ecx
  00709  c1 e8 10  shr      eax, 16                 ; 00000010H
  0070c  5e                 pop      esi
  0070d  33 c1              xor      eax, ecx
  0070f  5b                 pop      ebx
; 840  : }
  00710  c3                 ret      0
?HashAlfalfa_DWORD@@YAIPBDK@Z ENDP                  ; HashAlfalfa_DWORD
?HashAlfalfa_HALF@@YAIPBDK@Z PROC                   ; HashAlfalfa_HALF
; 809  : {
  00720  53                 push     ebx
  00721  55                 push     ebp
; 810  : UINT hash = 12;
; 811  : UINT hashBUFFER;
```

```
; 812  : SIZE_T i;
; 813  : for(i = 0; i < (wrdlen & -4); i += 4) {
  00722  8b 6c 24 10        mov      ebp, DWORD PTR _wrdlen$[esp+4]
  00726  56                 push     esi
  00727  8b dd              mov      ebx, ebp
  00729  33 d2              xor      edx, edx
  0072b  83 e3 fc  and      ebx, -4                  ; fffffffcH
  0072e  57                 push     edi
  0072f  8b 7c 24 14        mov      edi, DWORD PTR _key$[esp+12]
  00733  b9 0c 00 00 00     mov      ecx, 12                  ; 0000000cH
  00738  76 44              jbe      SHORT $LN4@HashAlfalf@3
  0073a  8d 9b 00 00 00
         00                 npad     6
$LL6@HashAlfalf@3:
; 814  :          //hash = (( ((hash<<5)-hash) + key[i] )<<5) - ( ((hash<<5)-hash) + key[i] ) + (key[i+1]);
; 815  :          hashBUFFER = ((hash<<5)-hash) + key[i];
; 816  :          hash = (( hashBUFFER )<<5) - ( hashBUFFER ) + (key[i+1]);
; 817  :          //hash = (( ((hash<<5)-hash) + key[i+2] )<<5) - ( ((hash<<5)-hash) + key[i+2] ) + (key[i+3]);
; 818  :          hashBUFFER = ((hash<<5)-hash) + key[i+2];
; 819  :          hash = (( hashBUFFER )<<5) - ( hashBUFFER ) + (key[i+3]);
  00740  0f be 34 17        movsx    esi, BYTE PTR [edi+edx]
  00744  8b c1              mov      eax, ecx
  00746  c1 e0 05  shl      eax, 5
  00749  2b c1              sub      eax, ecx
  0074b  0f be 4c 17 01     movsx    ecx, BYTE PTR [edi+edx+1]
  00750  03 f0              add      esi, eax
  00752  8b c6              mov      eax, esi
  00754  c1 e0 05  shl      eax, 5
  00757  2b c6              sub      eax, esi
  00759  03 c1              add      eax, ecx
  0075b  8b c8              mov      ecx, eax
  0075d  c1 e1 05  shl      ecx, 5
  00760  2b c8              sub      ecx, eax
  00762  0f be 44 17 02     movsx    eax, BYTE PTR [edi+edx+2]
  00767  03 c8              add      ecx, eax
  00769  8b c1              mov      eax, ecx
  0076b  c1 e0 05  shl      eax, 5
  0076e  2b c1              sub      eax, ecx
  00770  0f be 4c 17 03     movsx    ecx, BYTE PTR [edi+edx+3]
  00775  83 c2 04  add      edx, 4
  00778  03 c8              add      ecx, eax
  0077a  3b d3              cmp      edx, ebx
  0077c  72 c2              jb       SHORT $LL6@HashAlfalf@3
$LN4@HashAlfalf@3:
; 820  : }
; 821  : for(SIZE_T j = 0; j < (wrdlen & 3); j += 1) {
  0077e  33 c0              xor      eax, eax
  00780  83 e5 03  and      ebp, 3
  00783  8b f5              mov      esi, ebp
  00785  76 1d              jbe      SHORT $LN1@HashAlfalf@3
  00787  03 d7              add      edx, edi
  00789  8d a4 24 00 00
         00 00              npad     7
$LL3@HashAlfalf@3:
; 822  :          hash = ((hash<<5)-hash) + key[i+j];
  00790  0f be 3c 02        movsx    edi, BYTE PTR [edx+eax]
  00794  8b d9              mov      ebx, ecx
  00796  c1 e3 05  shl      ebx, 5
  00799  2b d9              sub      ebx, ecx
  0079b  03 fb              add      edi, ebx
  0079d  40                 inc      eax
  0079e  8b cf              mov      ecx, edi
```

```
  007a0  3b c6              cmp      eax, esi
  007a2  72 ec              jb       SHORT $LL3@HashAlfalf@3
$LN1@HashAlfalf@3:
  007a4  5f                 pop      edi
  007a5  5e                 pop      esi
; 823  : }
; 824  : return hash ^ (hash >> 16);
  007a6  8b c1              mov      eax, ecx
  007a8  c1 e8 10  shr      eax, 16                 ; 00000010H
  007ab  5d                 pop      ebp
  007ac  33 c1              xor      eax, ecx
  007ae  5b                 pop      ebx
; 825  : }
  007af  c3                 ret      0
?HashAlfalfa_HALF@@YAIPBDK@Z ENDP                   ; HashAlfalfa_HALF
?HashAlfalfa@@YAIPBDK@Z PROC                                ; HashAlfalfa
; 798  : {
  007b0  53                 push     ebx
; 799  : UINT hash = 7;
; 800  : for (SIZE_T i = 0; i < (wrdlen & -2); i += 2) {
  007b1  8b 5c 24 0c        mov      ebx, DWORD PTR _wrdlen$[esp]
  007b5  8b cb              mov      ecx, ebx
  007b7  83 e1 fe  and      ecx, -2                 ; fffffffeH
  007ba  56                 push     esi
  007bb  8b 74 24 0c        mov      esi, DWORD PTR _key$[esp+4]
  007bf  ba 07 00 00 00     mov      edx, 7
  007c4  76 24              jbe      SHORT $LN2@HashAlfalf@4
  007c6  49                 dec      ecx
  007c7  d1 e9              shr      ecx, 1
  007c9  8d 46 01  lea      eax, DWORD PTR [esi+1]
  007cc  41                 inc      ecx
  007cd  57                 push     edi
  007ce  8b ff              npad     2
$LL4@HashAlfalf@4:
; 801  :          hash = (53) * ((53) * hash + (key[i])) + (key[i+1]);
  007d0  0f be 78 ff        movsx    edi, BYTE PTR [eax-1]
  007d4  6b d2 35  imul     edx, 53                 ; 00000035H
  007d7  03 fa              add      edi, edx
  007d9  0f be 10  movsx    edx, BYTE PTR [eax]
  007dc  6b ff 35  imul     edi, 53                 ; 00000035H
  007df  03 d7              add      edx, edi
  007e1  83 c0 02  add      eax, 2
  007e4  83 e9 01  sub      ecx, 1
  007e7  75 e7              jne      SHORT $LL4@HashAlfalf@4
  007e9  5f                 pop      edi
$LN2@HashAlfalf@4:
; 802  : }
; 803  : if (wrdlen & 1)
  007ea  f6 c3 01  test     bl, 1
  007ed  74 0a              je       SHORT $LN1@HashAlfalf@4
; 804  :          hash = (53) * hash + (key[wrdlen-1]);
  007ef  0f be 44 1e ff     movsx    eax, BYTE PTR [esi+ebx-1]
  007f4  6b d2 35  imul     edx, 53                 ; 00000035H
  007f7  03 d0              add      edx, eax
$LN1@HashAlfalf@4:
; 805  : return hash ^ (hash >> 16);
  007f9  8b c2              mov      eax, edx
  007fb  c1 e8 10  shr      eax, 16                 ; 00000010H
  007fe  5e                 pop      esi
  007ff  33 c2              xor      eax, edx
  00801  5b                 pop      ebx
; 806  : }
```

```
  00802  c3                ret      0
?HashAlfalfa@@YAIPBDK@Z ENDP                                ; HashAlfalfa
?FNV1A_Hash_Jester@@YAIPBDK@Z PROC                   ; FNV1A_Hash_Jester
; 957  : const UINT PRIME = 709607;
; 958  : UINT hash32 = 2166136261;
; 959  : const char *p = str;
; 960  :
; 961  : // Idea comes from Igor Pavlov's 7zCRC, thanks.
; 962  : /*
; 963  : for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
; 964  :          hash32 = (hash32 ^ *p) * PRIME;
; 965  : }
; 966  : */
; 967  : for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
  004a0  8b 54 24 08        mov      edx, DWORD PTR _wrdlen$[esp-4]
  004a4  8b 44 24 04        mov      eax, DWORD PTR _str$[esp-4]
  004a8  56                 push     esi
  004a9  b9 c5 9d 1c 81     mov      ecx, -2128831035 ; 811c9dc5H
  004ae  83 fa 08  cmp      edx, 8
  004b1  72 2e              jb       SHORT $LN4@FNV1A_Hash
  004b3  8b f2              mov      esi, edx
  004b5  c1 ee 03  shr      esi, 3
  004b8  57                 push     edi
  004b9  8d a4 24 00 00
         00 00              npad     7
$LL6@FNV1A_Hash:
; 968  :          hash32 = (hash32 ^ *(DWORD *)p) * PRIME;
  004c0  8b 38              mov      edi, DWORD PTR [eax]
  004c2  33 f9              xor      edi, ecx
  004c4  69 ff e7 d3 0a
         00                 imul     edi, 709607              ; 000ad3e7H
; 969  :          hash32 = (hash32 ^ *(DWORD *)(p+4)) * PRIME;
  004ca  33 78 04  xor      edi, DWORD PTR [eax+4]
  004cd  83 ea 08  sub      edx, 8
  004d0  69 ff e7 d3 0a
         00                 imul     edi, 709607              ; 000ad3e7H
  004d6  83 c0 08  add      eax, 8
  004d9  83 ee 01  sub      esi, 1
  004dc  8b cf              mov      ecx, edi
  004de  75 e0              jne      SHORT $LL6@FNV1A_Hash
  004e0  5f                 pop      edi
$LN4@FNV1A_Hash:
; 970  : }
; 971  : // Cases: 0,1,2,3,4,5,6,7
; 972  : if (wrdlen & sizeof(DWORD)) {
  004e1  f6 c2 04  test     dl, 4
  004e4  74 0f              je       SHORT $LN3@FNV1A_Hash
; 973  :          hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
  004e6  8b 30              mov      esi, DWORD PTR [eax]
  004e8  33 f1              xor      esi, ecx
  004ea  69 f6 e7 d3 0a
         00                 imul     esi, 709607              ; 000ad3e7H
  004f0  8b ce              mov      ecx, esi
; 974  :          p += sizeof(DWORD);
  004f2  83 c0 04  add      eax, 4
$LN3@FNV1A_Hash:
; 975  : }
; 976  : if (wrdlen & sizeof(WORD)) {
  004f5  f6 c2 02  test     dl, 2
  004f8  74 10              je       SHORT $LN2@FNV1A_Hash
; 977  :          hash32 = (hash32 ^ *(WORD*)p) * PRIME;
  004fa  0f b7 30  movzx    esi, WORD PTR [eax]
```

```
  004fd  33 f1              xor      esi, ecx
  004ff  69 f6 e7 d3 0a
         00                 imul     esi, 709607              ; 000ad3e7H
  00505  8b ce              mov      ecx, esi
; 978  :          p += sizeof(WORD);
  00507  83 c0 02  add      eax, 2
$LN2@FNV1A_Hash:
  0050a  5e                 pop      esi
; 979  : }
; 980  : if (wrdlen & 1)
  0050b  f6 c2 01  test     dl, 1
  0050e  74 0d              je       SHORT $LN1@FNV1A_Hash
; 981  :          hash32 = (hash32 ^ *p) * PRIME;
  00510  0f be 00  movsx    eax, BYTE PTR [eax]
  00513  33 c1              xor      eax, ecx
  00515  69 c0 e7 d3 0a
         00                 imul     eax, 709607              ; 000ad3e7H
  0051b  8b c8              mov      ecx, eax
$LN1@FNV1A_Hash:
; 982  :
; 983  : return hash32 ^ (hash32 >> 16);
  0051d  8b c1              mov      eax, ecx
  0051f  c1 e8 10  shr      eax, 16                  ; 00000010H
  00522  33 c1              xor      eax, ecx
; 984  : }
  00524  c3                 ret      0
?FNV1A_Hash_Jester@@YAIPBDK@Z ENDP                   ; FNV1A_Hash_Jester
?FNV1A_Hash_Jesteress@@YAIPBDK@Z PROC                        ; FNV1A_Hash_Jesteress
; 989  : const UINT PRIME = 709607;
; 990  : UINT hash32 = 2166136261;
; 991  : const char *p = str;
; 992  :
; 993  : // Idea comes from Igor Pavlov's 7zCRC, thanks.
; 994  : /*
; 995  : for(; wrdlen && ((unsigned)(ptrdiff_t)p&3); wrdlen -= 1, p++) {
; 996  :          hash32 = (hash32 ^ *p) * PRIME;
; 997  : }
; 998  : */
; 999  : for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
  004a0  8b 54 24 08        mov      edx, DWORD PTR _wrdlen$[esp-4]
  004a4  8b 44 24 04        mov      eax, DWORD PTR _str$[esp-4]
  004a8  56                 push     esi
  004a9  b9 c5 9d 1c 81     mov      ecx, -2128831035 ; 811c9dc5H
  004ae  83 fa 08  cmp      edx, 8
  004b1  72 2b              jb       SHORT $LN4@FNV1A_Hash
  004b3  8b f2              mov      esi, edx
  004b5  c1 ee 03  shr      esi, 3
  004b8  57                 push     edi
  004b9  8d a4 24 00 00
         00 00              npad     7
$LL6@FNV1A_Hash:
; 1000 :          hash32 = (hash32 ^ (ROL(*(DWORD *)p,5)^*(DWORD *)(p+4))) * PRIME;
  004c0  8b 38              mov      edi, DWORD PTR [eax]
  004c2  c1 c7 05  rol      edi, 5
  004c5  33 78 04  xor      edi, DWORD PTR [eax+4]
  004c8  83 ea 08  sub      edx, 8
  004cb  33 f9              xor      edi, ecx
  004cd  69 ff e7 d3 0a
         00                 imul     edi, 709607              ; 000ad3e7H
  004d3  83 c0 08  add      eax, 8
  004d6  83 ee 01  sub      esi, 1
  004d9  8b cf              mov      ecx, edi
```

```
  004db  75 e3              jne      SHORT $LL6@FNV1A_Hash
  004dd  5f                 pop      edi
$LN4@FNV1A_Hash:
; 1001 : }
; 1002 : // Cases: 0,1,2,3,4,5,6,7
; 1003 : if (wrdlen & sizeof(DWORD)) {
  004de  f6 c2 04  test     dl, 4
  004e1  74 0f              je       SHORT $LN3@FNV1A_Hash
; 1004 :          hash32 = (hash32 ^ *(DWORD*)p) * PRIME;
  004e3  8b 30              mov      esi, DWORD PTR [eax]
  004e5  33 f1              xor      esi, ecx
  004e7  69 f6 e7 d3 0a
         00                 imul     esi, 709607              ; 000ad3e7H
  004ed  8b ce              mov      ecx, esi
; 1005 :          p += sizeof(DWORD);
  004ef  83 c0 04  add      eax, 4
$LN3@FNV1A_Hash:
; 1006 : }
; 1007 : if (wrdlen & sizeof(WORD)) {
  004f2  f6 c2 02  test     dl, 2
  004f5  74 10              je       SHORT $LN2@FNV1A_Hash
; 1008 :          hash32 = (hash32 ^ *(WORD*)p) * PRIME;
  004f7  0f b7 30  movzx    esi, WORD PTR [eax]
  004fa  33 f1              xor      esi, ecx
  004fc  69 f6 e7 d3 0a
         00                 imul     esi, 709607              ; 000ad3e7H
  00502  8b ce              mov      ecx, esi
; 1009 :          p += sizeof(WORD);
  00504  83 c0 02  add      eax, 2
$LN2@FNV1A_Hash:
  00507  5e                 pop      esi
; 1010 : }
; 1011 : if (wrdlen & 1)
  00508  f6 c2 01  test     dl, 1
  0050b  74 0d              je       SHORT $LN1@FNV1A_Hash
; 1012 :          hash32 = (hash32 ^ *p) * PRIME;
  0050d  0f be 00  movsx    eax, BYTE PTR [eax]
  00510  33 c1              xor      eax, ecx
  00512  69 c0 e7 d3 0a
         00                 imul     eax, 709607              ; 000ad3e7H
  00518  8b c8              mov      ecx, eax
$LN1@FNV1A_Hash:
; 1013 :
; 1014 : return hash32 ^ (hash32 >> 16);
  0051a  8b c1              mov      eax, ecx
  0051c  c1 e8 10  shr      eax, 16                  ; 00000010H
  0051f  33 c1              xor      eax, ecx
; 1015 : }
  00521  c3                 ret      0
?FNV1A_Hash_Jesteress@@YAIPBDK@Z ENDP                          ; FNV1A_Hash_Jesteress
?HashAlfalfa_Rollick@@YAIPBDK@Z PROC                 ; HashAlfalfa_Rollick
; 799  : UINT hash = 7;
; 800  : const char *p = key;
  00930  8b 4c 24 04        mov      ecx, DWORD PTR _key$[esp-4]
  00934  56                 push     esi
; 801  :
; 802  : for(; wrdlen >= 2*sizeof(DWORD); wrdlen -= 2*sizeof(DWORD), p += 2*sizeof(DWORD)) {
  00935  8b 74 24 0c        mov      esi, DWORD PTR _wrdlen$[esp]
  00939  ba 07 00 00 00     mov      edx, 7
  0093e  83 fe 08  cmp      esi, 8
  00941  72 54              jb       SHORT $LN4@HashAlfalf@5
  00943  53                 push     ebx
```

```
  00944  57                push     edi
  00945  8b fe             mov      edi, esi
  00947  c1 ef 03  shr      edi, 3
  0094a  8d 9b 00 00 00
         00                npad     6
$LL6@HashAlfalf@5:
; 803  :          DWORD x = (ROL(*(DWORD *)p,5)^*(DWORD *)(p+4));
  00950  8b 01             mov      eax, DWORD PTR [ecx]
; 804  :          hash = 53 * ( 53 * hash + ((x>>0)&0xFF) ) + ((x>>8)&0xFF);
; 805  :          hash = 53 * ( 53 * hash + ((x>>16)&0xFF) ) + ((x>>24)&0xFF);
  00952  6b d2 35  imul     edx, 53                 ; 00000035H
  00955  c1 c0 05  rol      eax, 5
  00958  33 41 04  xor      eax, DWORD PTR [ecx+4]
  0095b  83 ee 08  sub      esi, 8
  0095e  0f b6 d8  movzx    ebx, al
  00961  03 da             add      ebx, edx
  00963  6b db 35  imul     ebx, 53                 ; 00000035H
  00966  8b d0             mov      edx, eax
  00968  c1 ea 08  shr      edx, 8
  0096b  81 e2 ff 00 00
         00                and      edx, 255        ; 000000ffH
  00971  03 da             add      ebx, edx
  00973  6b db 35  imul     ebx, 53                 ; 00000035H
  00976  8b d0             mov      edx, eax
  00978  c1 ea 10  shr      edx, 16                 ; 00000010H
  0097b  81 e2 ff 00 00
         00                and      edx, 255        ; 000000ffH
  00981  03 da             add      ebx, edx
  00983  6b db 35  imul     ebx, 53                 ; 00000035H
  00986  c1 e8 18  shr      eax, 24                 ; 00000018H
  00989  03 d8             add      ebx, eax
  0098b  83 c1 08  add      ecx, 8
  0098e  83 ef 01  sub      edi, 1
  00991  8b d3             mov      edx, ebx
  00993  75 bb             jne      SHORT $LL6@HashAlfalf@5
  00995  5f                pop      edi
  00996  5b                pop      ebx
$LN4@HashAlfalf@5:
; 806  : }
; 807  : for(; wrdlen; wrdlen -= 1, p++) {
  00997  85 f6             test     esi, esi
  00999  74 13             je       SHORT $LN1@HashAlfalf@5
  0099b  eb 03 8d 49 00    npad     5
$LL3@HashAlfalf@5:
; 808  : hash = 53 * hash + *p;
  009a0  0f be 01  movsx    eax, BYTE PTR [ecx]
  009a3  6b d2 35  imul     edx, 53                 ; 00000035H
  009a6  4e                dec      esi
  009a7  03 d0             add      edx, eax
  009a9  41                inc      ecx
  009aa  85 f6             test     esi, esi
  009ac  75 f2             jne      SHORT $LL3@HashAlfalf@5
$LN1@HashAlfalf@5:
; 809  : }
; 810  : return hash ^ (hash >> 16);
  009ae  8b c2             mov      eax, edx
  009b0  c1 e8 10  shr      eax, 16                 ; 00000010H
  009b3  33 c2             xor      eax, edx
  009b5  5e                pop      esi
; 811  : }
  009b6  c3                ret      0
?HashAlfalfa_Rollick@@YAIPBDK@Z ENDP                 ; HashAlfalfa_Rollick
```

16